Anonymous Networked Group Communication: A Review and Meeting System Design

Gokhan Demirel, Gozde Ayranci and Oznur Ozkasap

Abstract—**Anonymity feature is significant for several networked group applications when the participants need to communicate while keeping their identities secret. We consider anonymous protocols used on the Internet and develop a prototype system, Anonymous Online Meeting, for anonymous group communication in which participants' identities are kept in secret inside the group as well as to other third parties. The system allows a group of users to participate in the meeting and transmit text and audio messages. We first give a review of anonymous protocols. Then, we describe the protocol used in our system, which is inspired by P^5 (Peer-to-Peer Personal Privacy Protocol) and discuss its design details.**

Index Terms—**Anonymity, Anonymous Protocols, Networked Group Communication, Online Meeting, Peer-to-Peer Personal Privacy Protocol (P^5)**

I. Introduction

ALTHOUGH encryption, the methodology for concealing information, is being used in network applications for secure communication, it does not hide the identity of communicating end systems. However, the identity information of end systems captured can be used to one's disadvantage. Protecting this information and hence providing a level of anonymity, in the context of various applications would be beneficial. Various levels of anonymity, namely sender anonymity, receiver anonymity, and sender-receiver unlinkability, are possible. From an observer's point of view, these are attained if she cannot identify the sender, identify the receiver, or link the sender to the receiver, respectively.

In this study, we consider the issue of anonymous group communication and first review anonymous protocols used on the Internet. Then, we describe the design and implementation details our software system, namely Anonymous Online Meeting (AOM), that allows a group of users to participate in an anonymous meeting by sending and receiving text and audio messages. AOM is basically a client-server application in which clients must authenticate themselves with the server in order to participate in a group meeting. Server only participates in the authentication phase, after the meeting is initiated, communication between clients continues in a peer-to-peer manner. The key point about the communication environment is that the information on who is connected to meeting is available to all participants in the group. However, it is not known which participant has sent which message.

The AOM system is inspired by and relies on the Naive Solution described in P^5 protocol, which provides strong anonymity and efficient communication for a small group of users [1]. A useful application of our system can be having discussion between, for example, a manager and her employees, in which none of the employees wanted the manager to know who said what. Although the manager maybe able to guess, she will not have a way of conclusively proving it (i.e. by performing some kind of IP trace). The analysis with network analyzer tools shows that the intended anonymity for communication between users without divulging the identity of who said what is achieved. In addition to providing anonymity, the prototype achieves live communication between users with text and audio messages.

II. Anonymous Protocols

In this section, we review anonymous protocols and describe the design choice for the approach that we used in AOM.

The Dining Cryptographers (DCnet) protocol provides sender anonymity under an adversary model [2]. DC-Net assumes a public key infrastructure, and users send encrypted broadcasts to the entire group, thus achieving receiver anonymity. However, all members of the group are made aware of when a message is sent, so Dining Cryptographers does not have high level of sender-receiver anonymity. Also, in DC-net, only one user can send at a time, so it is not suitable for an online live communication environment where multiple users should be able to send messages at a time.

Another study [5] has defined a method to let users anonymously send e-mail to the receivers using nodes that are called "Mixes". A Mix receives pieces of encrypted e-mail from many sources, it holds, re-sorts, may introduce null mail, hides sender by rewriting the from address, and it may send this processed e-mail to another mix for delivery to receiver. The anonymity that aimed here is each piece of resulted e-mail

Manuscript received September 24, 2007.

G. Demirel was with the Nortel Networks Turkey, Istanbul, 34768 Turkey. He is now with the Ericsson Turkey, Istanbul, 34398 Turkey (e-mail: gokhan.demirel@ericsson.com).

G. Ayranci, is with the Turkcell Communication Services PLC, Istanbul, 81440 Turkey. (e-mail: gozde.ayranci@turkcell.com.tr).

O. Ozkasap is with the Computer Engineering Department, Koc University, Istanbul, 34450 Turkey (e-mail: oozkasap@ku.edu.tr).

is equally likely to have come from any original sender.

Crowds is an anonymous protocol for web-transactions [4]. This protocol involves a group of users, called a crowd, each wants to communicate with a corresponding web server but without revealing his identity. The idea is to randomly route each message through the crowd until one member of the crowd decides to pass it to the server. This ensures that neither the receiver nor the nodes in the system can tell who sent the message. This system requires all nodes to be connected to all other nodes, and so it scales badly to larger networks.

The idea behind Hordes is based on Crowds work. Instead of serving as a proxy for web-transactions, random Mixes, as described in Mixes protocol, serve as proxies for UDP connections [3]. As in the case of Crowds, Hordes provides sender anonymity. However, it does not provide receiver anonymity, because as at some point the last random Mix that decides to pass the data to the end receiver makes a connection with the receiver. On the other hand, sender and receiver cannot be linked if the sender does not betray personal information to the receiver.

Peer-to-Peer Personal Privacy Protocol (P^5) is based on pure broadcast [1]. P^5 provides sender anonymity, receiver anonymity, and sender-receiver unlinkability. It is designed to be implemented over the current Internet protocols, and does not require any special infrastructure support.

Naïve Solution on P^5 is based on the following broadcast scenario: Consider a global broadcast channel that all participants in the anonymous communication send fixed length packets onto this channel at a fixed rate. These packets are encrypted such that only the recipient of the message may decrypt the packet. Assume that there is a mechanism to hide, spoof, or re-write sender addresses, e.g., by implementing the broadcast using an application-layer P2P ring, and that all messages are sent to the entire group. Lastly, every message is hop-by-hop encrypted, and thus, it is not possible to map a specific incoming message to a node to a particular outgoing message. It is possible that a node may not be actively communicating at any given time, but in order to maintain the fixed communication rate, it would have to send a packet anyway.

The communication environment described above provides receiver anonymity, since the sender does not know where in the broadcast group the receiver is or which host or address the receiver is using; the sender only knows that the receiver is part of the broadcast group. This system also provides sender anonymity, since all messages to a given receiver (in case of a ring) come from a single upstream node, and the receiver cannot determine the original sender of a message. Lastly, this solution also provides unlinkability from a passive adversary since the adversary is not able to gain any extra information from monitoring any (or all) network links.

Due to its broadcast nature, the naive solution to anonymity is not scalable. As the number of communicating people in the channel increases, the available bandwidth for any useful communication will decrease linearly since all messages are broadcasted to the communication channel. In addition, end-to-end reliability will decrease exponentially. P^5, being enhanced version of naïve solution, is based upon this basic broadcast channel principle; and scalability is achieved by creating a hierarchy of broadcast channels.

Our prototype system, AOM, is based on the naive solution described in P^5 protocol to achieve anonymity. P^5 is designed for a group to communicate online anonymously and easy to implement. P^5 has advantageous properties supported together such as sender anonymity, receiver anonymity and sender-receiver unlinkability. In addition to the naive solution part, we developed our prototype to be applicable to a message broadcast. The main purpose of our system is that the anonymous sender will send its message to all participants of the meeting. Because of this, in our implementation every participant is able to read each message broadcast, but the sender of message is kept secret.

III. AOM Prototype System

In this section, we present AOM system details and design decisions. The communication among participants is provided by datagram packet transportation over UDP. System details on Anonymity, Authentication, Message Structures, Handling Packets, and Audio Communication are described.

A. Providing Anonymity

In order to achieve the anonymity, the requirements are as follows:

1. All nodes are arranged on a logical ring - upstream and downstream neighbors are going to be known but this knowledge cannot be used to map their identities.
2. Messages should not contain any information that distinguishes original sender from last hop forwarder
3. All messages should be fixed length
4. All messages should be per hop encrypted
5. All nodes should send noise packets - a participant may have nothing to say, however it creates a “noise” packet and sends it to upstream node as if it is a real packet.

The requirements 1-4 provide sender anonymity and receiver anonymity. Sender anonymity is provided since the receiver of the message knows that the message comes from downstream node but it cannot map the message to the original sender. Receiver anonymity is provided since all messages go to all other nodes in a ring fashion. Therefore, an outside observer cannot distinguish that whether all the nodes talking to each other or the message initiated by a sender goes to a specific receiver. Requirement 5 provides sender and receiver unlinkability since again an outside observer cannot distinguish whether two specific nodes are talking to each other or the group is communicating as a whole.

In our prototype, we implemented all the five requirements. First of all, we created the application-layer logical ring at the authentication phase of the meeting. For this purpose, the server randomly assigns integer indexes to each client as they

authenticate themselves to server by providing valid meeting id and password, and orders client nodes according to their indexes. After that, it assigns $(i+1)^{th}$ node to i^{th} node as an upstream node. The meeting begins when all participants of the meeting have authenticated with the server. Therefore, after the authentication, the server has the logical ring structure for the meeting and it sends each node the address and public key (required for hop by hop encryption) of the node's upstream node. Through assigning random indexes to nodes, the server creates the logical ring independent of the order which nodes login.

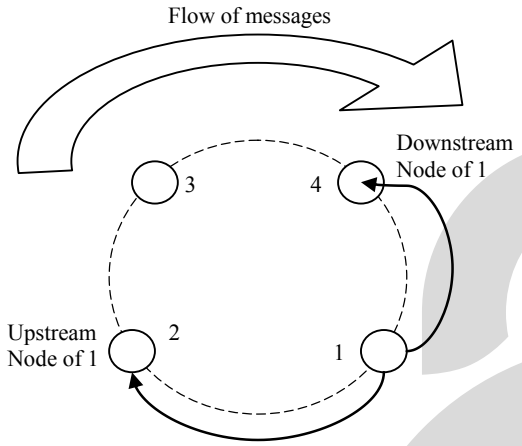

Fig. 1. Application-layer logical ring structure

The clients send messages on the logical ring as follows:

Suppose, node 1 in Fig. 1 wants to send a message m with type T length L and time stamp S to the other clients:

- It has node 2's public key provided by server before meeting starts
- It generates a session key Ks_1
- It encrypts [T || L|| Ks_1 || S] by public key of node 2, PU_{Node2}
- It encrypts [m] by Ks_1
- It encrypts [S] by its own public key PU_{Node1}
- It sends the message (PU_{Node2}[T || L|| Ks_1 || S]) || (Ks_1[m]) || (PU_{Node1}[S]) to node 2
- Each node forwards each incoming packet, so, node 2 decrypts the message and generates a
- new session key Ks_2 and the message
- (PU_{Node3}[T || L|| Ks_2 || S]) || (Ks_2[m]) || (PU_{Node1}[S]) is send to the node 3

where || denotes concatenation, PU_A denotes RSA public key of A, PU_A[m] denotes RSA public key encryption of m, Ks denotes DES secret key used as per hop session key and Ks[m] denotes DES encryption of message m using Ks.

In the above scheme, the last part PU_{Node1}[S] which is concatenated to the message functions as a signature of the node. This signature is used to prevent infinite looping of packets, upon receiving a message each node checks this signature basically trying to decrypt the signature, if it can decrypt the signature, node can determine that the whole message is generated by itself and does not forward this message anymore. This does not conflict with the second requirement above, since any node cannot get any information about the original sender of the message, each node only gets the information that this message is not generated by itself initially.

The Naïve Solution of P^5 requires a per hop encryption and it assumes that two nodes in a group want to communicate, for example say Node 1 and Node 3 want to communicate and Node 1 encrypts its message with public key of Node 3 before sending, and each node on the follow of the message encrypts the message with its upstream node's public key. However, in our system, all nodes should be able to read the incoming message. Therefore, the requirement for per hop encryption is achieved by session key generation in each node on the follow of the message and encrypting session key with upstream node's public key.

In AOM, all messages have the same fixed length. To provide fixed length messages, we padded the messages if the user wants to send a shorter text message than the fixed length packet payload. Besides, we introduce *noise* messages. Noise messages do not differ from any other message except their type being noise and when the type of the message is decrypted and retrieved, a noise message is forwarded and no processing is done after forwarding. Each node sends noise messages periodically. Of course noise packets reduce the efficient usage of network bandwidth, however they are introduced to provide sender and receiver unlinkability. Therefore, they should be used.

B. Authentication

Authentication of the clients is another important part in AOM. Before the start of each meeting, clients must authenticate themselves with the server in order to participate in the meeting. We introduce two phases for a meeting to start; first a client, who wants to initiate a meeting, sends a request to the server, server provides the client a meeting id and the specific password for that specific meeting. In the second phase, each client who wants to join to the meeting authenticates himself to the server by providing a meeting id and the meeting password for that meeting.

Initialization phase is as follows (see Fig. 2 for flow of messages):

- Client who wants to initialize a meeting send a request to the server
- Server, upon receiving request, sends its public key to the client
- Client generates a random nonce N_1, concatenates the desired number of attendees to it and finally generates and concatenates a DES secret key as a session key, encrypts the resulting stream with server's public key using RSA and sends it to the server
- Upon receiving the message from client, server generates a random meeting id and reserves it for # of attendees, generates a meeting password, and creates a message that

has received nonce N_1 concatenated to it meeting id, concatenated to it generated password length, concatenated to it meeting password and encrypts the resulting message with the session key sent by client. It adds the encrypted message's length at the beginning of the encrypted message and sends to the client

- When client receives this meeting data, it sends a message encrypted with server's public key and encrypted by RSA that has meeting id in it to the server

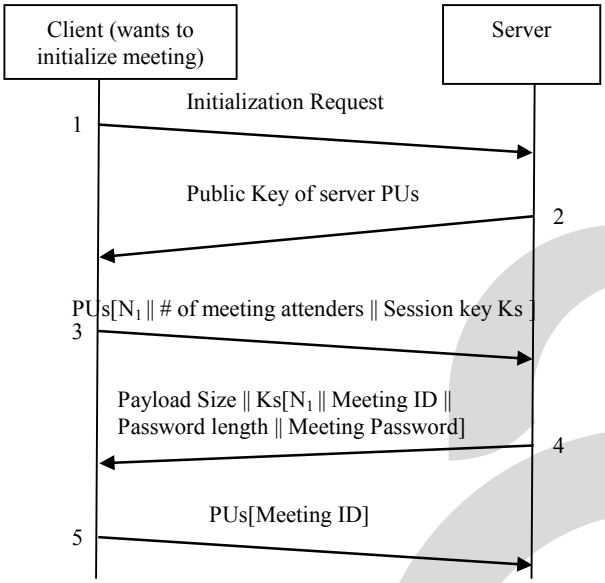

Fig. 2. Message flow for initialization of a meeting

In the scheme described above, nonce N_1 is used to prevent future replay attacks. In addition, the password generated as a meeting password by the server is between 8 or 16 characters in length and it includes random characters and digits. Each user who wants to join to a meeting must provide the meeting id and the meeting password so that we assume that the meeting id and the password of the meeting are provided to other users by the user who initialized the meeting. As the last action of the initialization phase, server, after receiving the 5th message in the Fig. 2, starts a timer for this reserved meeting and waits for join requests to this meeting. If all participants join before the timer expires, the meeting is started.

As the second phase, users join to the meeting (see Fig. 3 for flow of messages). In the join phase, the actual authentication of the users to the server is needed and provided by meeting id and the meeting password.

Join phase is as follows:

- Client who wants to initialize a meeting send a request to the server
- Server, upon receiving request, sends its public key to the client
- Client concatenates meeting id, password length, meeting password, its own RSA public key and encrypts those using RSA with server's public key then sends to the server
- Server checks the meeting id and password and returns a message indicating success or fail.
- If the password and meeting id provided by the user are true and if all the attendees join to the meeting, server creates the logical ring of nodes. Following that, server sends each node its upstream node's address and public key over a TCP connection to make sure that this information is transported reliably. If the time expires before all users attend, error messages are sent to users that are already joined over TCP connection again.

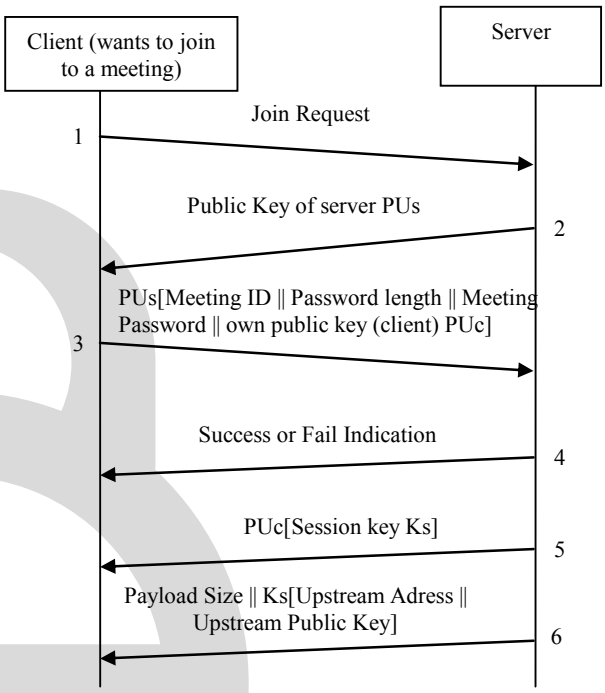

Fig. 3. Message flow for joining to a meeting

C. *Message Structures and Handling of Packets*

In our prototype, all packets received by the server are handled by a separate thread. A thread is devoted to listen to the socket for requests and when a packet is received, the packet is momentarily transferred to a thread to process it and the same thread sends the response. Therefore, the server is stateless and it is designed as thread-per-request multi-threaded server. On the other hand, the client software has devoted threads for initialization of a meeting and joining to a meeting since it needs to keep state in those phases. However, when client software communicates on the logical ring fashion, it also operates as multi-threaded. Again a thread listens for the incoming messages, and each received packet is transferred to a thread to handle it. On the communication phase of the client, there are also some threads executed to manage audio buffering and text buffering, repetitively sending noise packets and if the user is sending audio messages there is a thread to capture audio from the line in. We have the following message structures to achieve handling of all types of communications.

1. Message Structures used between Client and Server

A client who wants to initiate a meeting or join to a meeting sends messages to server indicating its request. The Initialization Request (see Fig. 2 message 1) and Join Request (Fig. 3 message 1) do not contain any data; they are messages that are identified by their types.

At the initialization phase, client who wants to initialize meeting should send a message to the server that indicates the number of attendees (Fig. 2 message 3). This message contains message type as to be identified by server and 256 byte RSA encrypted data. When decrypted, RSA encrypted data contains 5 byte nonce, 2 byte number of meeting attendees and 24 byte session key's byte representation.

As in the initialization phase, at the join phase, client who wants to join to a meeting should send a message to the server that indicates the meeting to be joined and corresponding password of that meeting (Fig. 3 message 3). This message contains message type and 256 byte RSA encrypted data. When decrypted, RSA encrypted data contains 3 byte meeting ID, 2 byte password length, meeting password field (size defined by previous field), 162 byte client's RSA public key's byte representation.

The 5th message in Fig. 2 is sent to the server when a client wants to end the meeting or a node dies and its upstream node detects this by timeout (the meeting should end in a dead node case since logical ring would be broken). This message contains message type and 256 byte RSA encrypted data. RSA encrypted data contains 3 byte meeting ID.

The 2nd messages in both Fig. 2 and Fig. 3 are messages that sent to the client after initialization and join request messages. They include message type field and 294 byte Server's RSA public key's byte representation. Upon receiving an initialization message, server should provide the client the meeting id and password. This is achieved by 3rd message in Fig. 2. This message has also its type concatenated to the beginning and 2 byte payload size field and DES encrypted meeting data (size defined by previous field). DES encrypted data contains 5 byte nonce, 3 byte meeting ID, 2 byte password length field, meeting password (size defined by previous field).

In the join phase, the success or fail indication at the 4th step in Fig. 3 is achieved by two messages in the protocol between client and server. The former is an OK message that indicates client's meeting id and password is correct and its join request is successful. The latter is an Error message to indicate that client's meeting id and password is incorrect and its join request is unsuccessful. Also, in initialization phase there is a message, similar to these messages, which is a Reject message. It is sent by server to client to indicate that there is no empty space for a new meeting and initialization request is rejected. These messages do not contain any data; they are identified by clients by their message types.

The last message used in between client and server is a message sent by the server to the client when meeting is about to start, it includes upstream node's address and public key, Fig. 3 message 6. This message contains message type field, 3 byte payload size field and DES encrypted meeting data (size defined by previous field). DES encrypted data contains 4 byte upstream node's IP address and 162 byte upstream node's RSA public key's byte representation.

2. Message Structures used in P2P Client Communication

There are four types of messages that are used in the P2P client communication. The first one is for text message transportation. This message contains RSA encrypted message header, DES encrypted payload, RSA encrypted signature. The message header part contains a type field to indicate it is a text message, which is TX in this case, a payload size field which defines useful data in payload, session key's byte representation, and timestamp. DES encrypted payload contains ASCII text message (size defined in payload size field), and ASCII padding. The signature contains timestamp which is not used; this field is just for checking signature (See section 3.A above for details). In fact, four types of messages that are used in the P2P client communication have this signature concatenated to the end of the message.

As the system provides audio communication, there is a message for audio data. It contains RSA encrypted message header, DES encrypted payload, RSA encrypted signature. When decrypted, message header contains message type field which is AU in this case, payload size field which is a fix size for audio messages, session key's byte representation, and timestamp. DES encrypted payload contains audio message data.

As described in previous sections, nodes should send noise messages. This is the 3rd message type used between client-to-client communication. It contains RSA encrypted message header, DES encrypted payload and RSA encrypted signature. Message header contains message type field which is NS in this case, payload size field which is not useful for noise messages, session key's byte representation, and timestamp. Payload contains fixed length noise message. Like text messages noise messages are forwarded, until they reach their original sender (See section 3.A above for details).

The last message that is used in client-to-client communication is a Stop message. Stop message contains again a RSA encrypted message header, DES encrypted payload, and signature. Message header contains message type field which is in this case ST, payload size field which is also not useful for stop messages, session key's byte representation, and timestamp. Payload contains DES encrypted noise. This message is used to stop meeting between clients. A meeting attendee may want to leave the meeting, however in this case he or she may not leave the meeting immediately because the logical ring would be broken and communication would end. Therefore, if a meeting attendee wants to leave the meeting, the stop message is sent to all clients through the logical ring and meeting automatically ends for all clients. In addition, if a node in the logical ring is dead, the meeting should end because of the same reason, so there is a timeout mechanism to

detect dead nodes. We know that each node sends a noise message in random amounts of time but the upper limit of this interval is known and a timeout interval is set according to this limit. If a node does not receive messages for a predetermined time, it times out and sends a Stop message to its upstream node, if the upstream nodes are still alive they will forward message through the logical ring and meeting and the communication will end as a result of the "Stop" message.

D. Audio Communication

An additional feature that AOM achieves is audio communication. If a user wants to send audio messages, the system initiates a thread for capturing audio and in each second an 8000 byte audio data is captured. We directly pass this data to UDP channel without any compression or fragmentation. As a future work, compression and fragmentation mechanisms can be implemented to lower the messages sizes.

IV. CONCLUSIONS AND FUTURE WORK

We describe our system, Anonymous Online Meeting, that provides strong anonymity for a group of communicating participants. We analyzed the message traffic of our prototype with the network analyzer tool Ethereal [6] and confirmed that the anonymity properties are provided. Each communicating node can only determine its upstream and downstream nodes however cannot map any message to a specific sender and an outside observer can only observe that a group of nodes are communicating but he or she cannot either map a message to its sender or a receiver or link a sender and a receiver. Another point is that all messages sent through the logical ring have the same size. Because of this, when a network analyzer is used, it is not possible to follow a packet's route by looking its size, and analyze its sender. Any packet may be initiated in any node, and no information can be got about the packet initiator. As a result, the prototype achieves sender anonymity, receiver anonymity and sender and receiver anonymity. As further study, the system can be improved to support efficient scalable anonymous online meetings that can be used by large group of communicating users.

REFERENCES

[1] Sherwood, R., Bhattacharjee, B., Srinivasan, A. "P^5: A Protocol for Scalable Anonymous Communication," In the Proceedings of the IEEE Symposium on Security and Privacy, 2002.

[2] A. Jones. (2004, September). Anonymous Communication on the Internet. Retrieved: September 21 2007, from http://www10.cs.rose-hulman.edu/Papers/Jones.pdf

[3] T. Chothia, K. Chatzikokolakis, "A Survey of Anonymous Peer-to-Peer File-Sharing," In the Proceedings of the IFIP International Symposium on Network-Centric Ubiquitous Systems (NCUS), Springer, LNCS 3823, pp. 744-755, 2005.

[4] Reiter, M., Rubin, A. "Crowds: Anonymity for Web Transactions," In ACM Transactions on Information and System Security, 1(1), pp.66-92, 1998.

[5] J. Claessens, B. Prenee, J. Vandewalle, "Solutions for Anonymous Communication on the Internet," In Proceedings of the IEEE 33rd International Carnahan Conference on Security Technology, 1999.

[6] Ethereal: A Network Protocol Analyzer http://www.ethereal.com/ Retrieved September 21, 2007